Towards Refinement Strategy Planning for Event-B

Tsutomu Kobayashi[1] and Shinichi Honiden[1][2]

[1] The University of Tokyo, Japan
[2] National Institute of Informatics, Japan
{t-kobayashi,honiden}@nii.ac.jp

Abstract. Event-B is a formal approach oriented to system modeling and analysis. It supports refinement mechanism that enables stepwise modeling and verification of a system. By using refinement, the complexity of verification can be spread and mitigated. In common development using Event-B, a specification written in a natural language is examined before modeling in order to plan the modeling and refinement strategy. After that, starting from a simple abstract model, concrete models in several different abstraction levels are constructed by gradually introducing complex structures and concepts. Although users of Event-B have to plan how to abstract the specification for the construction of each model, guidelines for such a planning have not been suggested. Specifically, some elements in a model often require that other elements are included in the model because of semantics constraints of Event-B. As such requirements introduces many elements at once, non-experts of Event-B often make refinement rough though rough refinement does not mitigate the complexity of verification well. In response to the problem, a method is proposed to plan what models are constructed in each abstraction level. The method calculates plans that mitigate the complexity well considering the semantics constraints of Event-B and the relationships between elements in a system.

Keywords: Formal Methods, Refinement, Event-B, Specifications, Abstraction

1 Introduction

Event-B [2] is a formal approach oriented to system-level analysis and modeling. Event-B users specify models in a notation based on set theory and first order logic and check the model by proving.

The most notable feature of Event-B is the support for refinement mechanism. In a refinement process, users first construct a simple and highly abstract model. After checking the consistency in the abstract model, more complex model is constructed and the consistency between the abstract model and the complex model is checked. Usually complex models are constructed by introducing new aspects and properties of a system. Users gradually construct concrete models by repeating this step. Refinement enables users to construct a complex model more simply rather than to construct the complex model at once. Therefore, the burden of a model construction is mitigated by Event-B.

As Event-B is an effective method, it has been attracting more and more attentions from the industry. For example, many companies in Japan are interested in formal methods including Event-B, while technical and administrative guidelines are constructed through cooperation of a national institute and software vendors [1].

Refinement enables users to spread the complexity of modeling over some steps. However, it is necessary to properly define how elements are gradually introduced, mitigating the complexity while complying with semantics constraints of Event-B. In this paper, we propose a method that considers the constraints and relationships between elements of a system and plans what models should be constructed for an effective refinement. Thus the method enables ordinary users to leverage the refinement mechanism in a simple way.

The remainder of this paper is organized as follows. Section 2 describes the problem and the cause of the problem. Section 3 describes the proposed method together with exemplification on an example. Section 4 shows related work as well as future work, before concluding remarks in Section 5.

2 Problem and Approach

In usual modeling in Event-B, users first read the specification of the system written in a natural language. Then models in several abstraction levels are constructed gradually. Event-B supports checking constructed models but does not guide modeling explicitly. Thus users have to plan what models are constructed in each abstraction level. Specification of a system is composed of a set of statements about property of the system. We call such statements *artifact*s. For example, a specification of a library management system may include an artifact "There are no loaned books in the open stack.". Usually, Event-B models include invariants that correspond to a subset of artifacts of the system. Thus, users have to plan which subset of artifacts of the system should be reflected to models of each abstraction level. When constructing a concrete model, new artifacts are added to artifacts of the abstract model. Therefore users need to plan which artifacts are introduced to each abstraction level.

In Event-B models, artifacts are expressed as invariants using the formal language of Event-B. Thus, in order to express an artifact, it is necessary to introduce elements (e.g. career sets, constants, variables, and events) that correspond to terms appeared in the artifact. We call such terms *phenomena*. For instance, an artifact "There are no loaned books in the open stack." can be expressed like "$openstack \cap \mathrm{dom}(loan\_state) = \emptyset$" (Where $openstack \subseteq books$ and $loan\_state \in books \rightarrow\!\!\!\!\!+ members$) only when *openstack* and *loan_state* are already introduced to the model.

Such constraints on the introduction of phenomena exist not only between an artifact and phenomena but also between a phenomenon and phenomena. That is, in some cases, an introduction of a phenomenon requires an introduction of some other phenomena. The causes of such constraints on introduction include the following two facts.

Firstly, all variables and constants in an Event-B model have to be typed as a primitive type (a built-in data type or an element of a career set as a user-defined type) or a pair that is recursively built from primitive types. A primitive type is atomic and not expressible by using other primitive types. For example, if a variable *var* is typed as S by using a career set S in an abstract model and typed as $T \to \mathbb{N}$ by using a career set T and a built-in data type $\mathbb{N}$ in a concrete model, a type error will occur. Thus, any typing statement of a phenomenon should not be changed through refinement. Therefore, a phenomenon corresponds to a variable or a constant can be introduced only when phenomena that are necessary to type the variable or constant are already introduced. For instance, in order to introduce a phenomenon "loan state" and express its type as $loan\_state \in books \nrightarrow members$, the introduction of career sets *books* and *members* is needed. Thus, it is also required to introduce phenomena that corresponds to career sets *books* and *members*.

Secondly, Event-B has several criteria for consistency between an abstract model and a concrete model. In modeling in Event-B, users can confirm consistency of models by proving them (proof obligations). There is a proof obligation named EQL, which requires that if a variable is included in both an abstract model and a concrete model there must not be a state transition such that it is included in the concrete model but not included in the abstract model. In order to make a refinement consistent EQL proof obligation must hold. Therefore a phenomenon corresponds to a variable can be introduced in a model only when state transitions that change the value of the variable are already introduced. For example, the introduction of a variable that corresponds to a phenomenon "loan state" requires an introduction of all state transitions included in the behavior of the system (e.g. "loaning a book from the open stack", "returning a book", "loaning a reserved book").

For these reasons, an introduction of an artifact to a model requires introductions of 1) phenomena that appear in the artifact and 2) phenomena that are required by 1). Let $A = (A_i)_{i=0,1,\cdots}(A_0 = \emptyset)$ be a sequence of sets of artifacts reflected to the nth model, $\text{req\_a}(a)$ be the phenomena required by an introduction of an artifact a, and $\text{req\_as}(A_i) = \bigcup_{a \in A_i} \text{req\_a}(a)$ for a set of artifacts A_i. When an artifact is introduced, phenomena that are not introduced yet but required for an introduction of the artifact are also introduced. Then, let $\text{int\_p}_i(A) = \text{req\_as}(A_i) \setminus \text{req\_as}(A_{i-1})$ $(1 < i)$[3]. It denotes newly installed phenomena to the ith model in a sequence of artifacts sets $(A_i)_{i=0,1,\cdots}$.

Consider the introduction order of artifacts a and b such that $\text{req\_a}(a) = \{p_1, \cdots, p_{10}\}$, $\text{req\_a}(b) = \{p_6, \cdots, p_{10}, q\}$ to an empty model (Figure 1). For $A_1 = \{a\}, A_2 = \{a, b\}$, the newly introduced phenomena will be as follows: $\text{int\_p}_1(A) = \text{req\_a}(a) = \{p_1, \cdots, p_{10}\}$, $\text{int\_p}_2(A) = \text{req\_a}(b) \setminus \text{req\_a}(a) = \{q\}$. In contrast, For $B_1 = \{b\}, B_2 = \{a, b\}$, the newly introduced phenomena will

[3] In this paper, the relative complement of a set T in a set S is denoted by $S \setminus T$.

be as follows: $\text{int\_p}_1(B) = \text{req\_a}(b) = \{p_6, \cdots, p_{10}, q\}$, $\text{int\_p}_2(B) = \text{req\_a}(a) \setminus \text{req\_a}(b) = \{p_1, \cdots, p_5\}$.

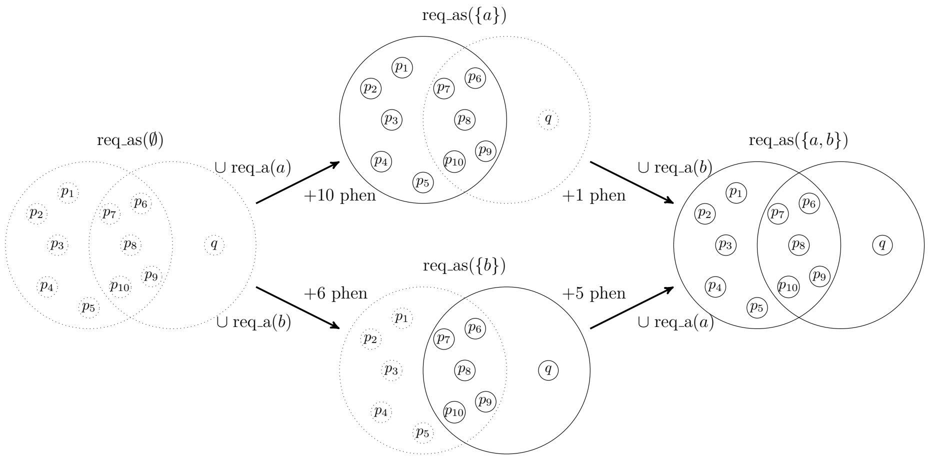

Fig. 1. Difference between Two Introduction Orders

Therefore, $(|\text{int\_p}_1(A)|, |\text{int\_p}_2(A)|) = (10, 1)$, whereas $(|\text{int\_p}_1(B)|, |\text{int\_p}_2(B)|) = (6, 5)$. Thus, the number of newly introduced phenomena in each model will vary depending on the introduction order of artifacts. As refinement is a mechanism to mitigate the complexity of modeling by spreading the introduction of phenomena over some steps, how much the numbers of newly introduced phenomena are dispersed is useful information for users to plan refinement. Thus, we define the *effectiveness* of an introduction order of artifacts as in Definition 1. For instance, the order (a, b) is more effective than (b, a) in the above example. Moreover, an order $(A_i)_{i=0,\cdots,3}$ such that $(num_{A,i})_{i=1,2,3} = (3, 1, 2)$ is more effective than an order $(B_i)_{i=0,\cdots,3}$ such that $(num_{B,i})_{i=1,2,3} = (0, 3, 3)$ since $(snum_{A,i})_{i=1,2,3} = (3, 2, 1)$ is smaller than $(snum_{B,i})_{i=1,2,3} = (3, 3, 0)$ in lexicographical order.

Definition 1. *Let a sequence $(num_{A,i})_{i=1,\cdots,|A|}$ be a history of the number of phenomena newly introduced in each refinement (i.e. $num_{A,i} = |\text{int\_p}_i(A)|$) and a sequence $(snum_{A,i})_{i=1,\cdots,|A|}$ be a sorted permutation of $(num_{A,i})_{i=1,\cdots,|A|}$ in descending order. Then, for a sequence $(A_i)_{i=0,\cdots,|A|}$ and $(B_i)_{i=0,\cdots,|B|}$ such that $\text{req\_as}(A_{|A|}) = \text{req\_as}(B_{|B|})$, $(A_i)_{i=0,\cdots,|A|}$ is called more effective than $(B_i)_{i=0,\cdots,|B|}$, if $(snum_{A,i})_{i=1,\cdots,|A|}$ is smaller than $(snum_{B,i})_{i=1,\cdots,|B|}$ in lexicographical order.*

As in the above example, users should plan an introduction order of artifacts so that the refinement is effective. However, for this planning, users have to grasp and compare the constraints on introduction between phenomena over multi-

Table 1. Artifacts of Library Management System

	Artifact	Phenomena Appeared in the Artifact
a	"Loan is done only for members"	loan state, members
b	"Books on loan are not in the open stack"	loan state, books, open stack state
c	"No reserved books are in the open stack"	reservation state, books, open stack state

Table 2. Events of Library Management System

	Event	Caused State Transitions
p1	Loaning a reserved books	Remove one from reservation state, Add one to loan state
p2	Returning a book	Remove one from loan state
p3	Loaning a book from the open stack	Remove one from open stack state, Add one to loan state

ple steps. The constraints are too complex for users to analyze in their heads. Therefore, Event-B users (especially beginners) have to repeat trial and error processes during modeling many times. Thus, though refinement is a powerful mechanism, it is not so easy to use refinement in realistic situations.

3 Method

3.1 Derivation of Required Phenomena

As we viewed in Section 2, the effectiveness of refinement depends on the sets of phenomena required by artifacts. The phenomena required by an artifact depend on types and state transitions related to phenomena that appear in the artifact. In the proposed method, constraints on introduction between phenomena related to types and state transitions are assumed as the input. The output of the method is orders that maximize the effectiveness of refinement.

To illustrate the method, construction of a model of a library management system is described. The artifacts of the system are as shown in Table 1 and the events of the system are as shown in Table 2. Events represent behavior of the system. An event can cause multiple state transitions.

Phenomena can be classified into four kinds according to what kind of element in an Event-B model corresponds to the phenomenon. Phenomenon corresponds to a career set, a constant, a variable, or an event.

Let P_{S}, P_{C}, P_{V}, and P_{E} be the set of phenomena that correspond to career sets, constants, variables, and events, respectively. Let $P = P_{\mathrm{S}} \cup P_{\mathrm{C}} \cup P_{\mathrm{V}} \cup P_{\mathrm{E}}$ and T be the set of state transitions.

Let $\mathit{typed} : P \to 2^{P_{\mathrm{S}}}$ be a set of career sets required for typing a constant or a variable, $\mathit{changed\_by} : P \to 2^{T}$ be a set of state transitions that change the value of a variable, and $\mathit{caused\_by} : T \to 2^{P_{\mathrm{E}}}$ be a set of events that causes a state transition.

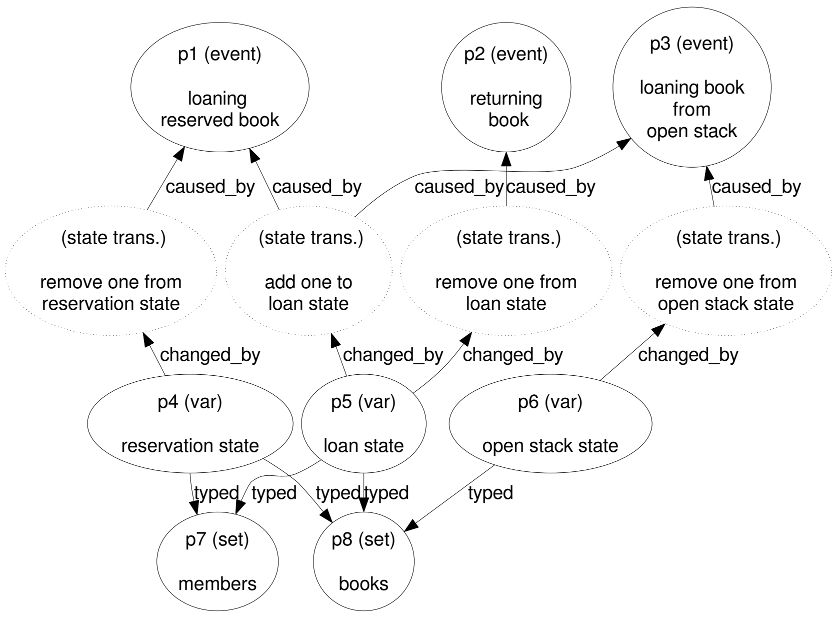

Fig. 2. Constraints on Introduction between Phenomena in Library Management System

The proposed method takes these three functions as the input from the users. For example, constraints on introduction between phenomena in the library management system can be depicted as in Figure 2.

The set of career sets required for typing a phenomenon can be derived by tracing *typed* edges. The set of events that change a phenomenon can be derived by tracing *changed_by* edges and *caused_by* edges. Therefore, let req_p(p) be the phenomena required by an introduction of a phenomenon p, then req_p(p) can be derived by the input. That is,

$$\text{req\_p}(p) = \mathit{typed}(p) \cup \bigcup_{t \in \mathit{changed\_by}(p)} \mathit{caused\_by}(t) \ . \tag{1}$$

Thus, req_a(a) can be derived by the input. That is,

$$\text{req\_a}(a) = \text{appear}(a) \cup \bigcup_{p \in \text{appear}(a)} \text{req\_p}(p) \ . \tag{2}$$

Where appear(a) denotes the phenomena that directly appear in an artifact a. By constraints on introduction depicted in Figure 2 and Equations (1) and (2), req_a of each artifact in the library management system is derived as follows:

$$\begin{aligned}
\text{req\_a}(a) &= \{\ \text{p1, p2, p3,} \quad\ \text{p5,} \quad\ \text{p7, p8}\ \} \\
\text{req\_a}(b) &= \{\ \text{p1, p2, p3,} \quad\ \text{p5, p6, p7, p8}\ \} \\
\text{req\_a}(c) &= \{\ \text{p1,} \quad\ \text{p3, p4,} \quad\ \text{p6, p7, p8}\ \}
\end{aligned}$$

Table 3. All Introduction Order of Artifacts in Library Management System

$(\text{int\_a}_i(A))_{i=1,2,3}$	$(\lvert\text{int\_p}_i(A)\rvert)_{i=1,2,3}$	Effectiveness Rank
$(\{a\},\{b\},\{c\})$	$(6,1,1)$	1
$(\{a\},\{c\},\{b\})$	$(6,2,0)$	2
$(\{c\},\{a\},\{b\})$	$(6,2,0)$	2
$(\{c\},\{b\},\{a\})$	$(6,2,0)$	2
$(\{b\},\{a\},\{c\})$	$(7,0,1)$	3
$(\{b\},\{c\},\{a\})$	$(7,1,0)$	3

3.2 Search for the Best Introduction Orders of Artifacts

Let $\text{int\_a}_i(A) = A_i \setminus A_{i-1}$ $(1 < i)$. It denotes newly introduced artifacts to the ith model. We assume that only one artifact is introduced through one refinement step. Then, $|\text{int\_a}_i(A)| = 1$ $(1 < i)$ and $\text{req\_as}(A_i) = \text{req\_a}(a) \cup \text{req\_as}(A_{i-1})$ $(1 < i)$ where $a = |\text{int\_a}_i(A)|$.

In the proposed method, orders that correspond to sequences $(A_i)_{i=0,\cdots,|A|}$ that maximized the effectiveness of the refinement are obtained by Algorithm 1. All orders of artifacts introduction $((\text{int\_a}_i(A))_{i=1,2,3})$ and the numbers of newly introduced phenomena $((|\text{int\_p}_i(A)|)_{i=1,2,3})$ in each refinement for the library management system is as shown in Table 3. In this case, the result of the algorithm represents the order $(\{a\},\{b\},\{c\})$.

The method uses breadth first search with pruning as shown in the Algorithm 1. A node of the search tree represents an introduction order of artifacts. A structure that represents a node is composed of *as* that represents the history of artifacts introduction, *ps* that represents the set of phenomena introduced so far, *nums* that represents the history of the number of introduced phenomena in each step, *max* that represents the maximum of *nums*, and *rest* that represents the number of phenomena not introduced yet.

The function CertainlyBetter (Line 1–19) checks whether an introduction order of artifacts is certainly effective than the other order. This function is used for pruning (Line 32, 35). The number of phenomena introduced in a later refinement is at most the number of phenomena not introduced yet (*maybe_better.rest*) since $\text{req\_as}(A_i) \subseteq \text{req\_as}(A_{|A|})$ for all i. Therefore, an order is certainly better if the maximum number of introduced phenomena in each step of the order is at most less than the current maximum of the other order (Line 5–6). If both maximums are equal, the algorithm retries the checking on orders without the artifact corresponds to the maximum number (Line 7–15).

Algorithm 1 Search the Best Introduction Orders of Artifacts

```
function CertainlyBetter($maybe\_better, maybe\_worse$)
    if $((maybe\_better.nums = \{\}) \vee (maybe\_worse.nums = \{\}))$ then
        return $false$                    ▷ Not sure whether $maybe\_better$ is better
    else if $\{max(\{maybe\_better.max, maybe\_better.rest\}) <$
            $maybe\_worse.max)$ then
        return $true$                          ▷ $maybe\_better$ is certainly better
    else if $maybe\_better.max = maybe\_worse.max$ then
        new $mb\_reduced, mw\_reduced$
        $mb\_reduced.nums \leftarrow maybe\_better.nums \setminus \{maybe\_better.max\}$
        $mw\_reduced.nums \leftarrow maybe\_worse.nums \setminus \{maybe\_worse.max\}$
        $mb\_reduced.max \leftarrow max(mb\_reduced.nums)$
        $mw\_reduced.max \leftarrow max(mw\_reduced.nums)$
        $mb\_reduced.rest \leftarrow maybe\_better.rest$
        $mw\_reduced.rest \leftarrow maybe\_worse.rest$
        return CertainlyBetter($mb\_reduced, mw\_reduced$)
    else
        return $false$                    ▷ Not sure whether $maybe\_better$ is better
    end if
end function

function SearchBestOrder($artifacts$, req_as, $n\_artif, n\_phen$)
    $orders \leftarrow \{\{as : \{\},\ ps : \emptyset,\ nums : \{\},\ max : 0,\ rest : n\_phen\}\}$
    repeat
        for all $order \in orders$ s.t. $length(order.as)$ is minimum do
            $orders \leftarrow orders \setminus \{order\}$
            for all $a \in (artifacts \setminus order.as)$ do
                new $neworder$
                $neworder.as \leftarrow append(order.as, \{a\})$
                $neworder.ps \leftarrow$ req_as$(neworder.as)$
                $neworder.nums \leftarrow append(order.nums, \{|neworder.ps \setminus order.ps|\})$
                $neworder.max \leftarrow max(neworder.nums)$
                $neworder.rest \leftarrow n\_phen - sum(neworder.nums)$
                if not $(\exists o \in orders$ s.t. CertainlyBetter$(o, neworder))$ then
                    $orders \leftarrow orders \cup \{neworder\}$
                    for all $o \in orders$ do
                        if CertainlyBetter$(neworder, o)$ then
                            $orders \leftarrow orders \setminus \{o\}$
                        end if
                    end for
                end if
            end for
        end for
    until $\forall o \in orders \,.\, length(o.as) = n\_artif$
    return $orders$
end function
```

4 Discussion

4.1 Related Work

There are some studies on requirement engineering methods for modeling in Event-B. In [9], Problem Frames and Event-B are applied successfully on an industrial project. The authors constructed a problem diagram before modeling in Event-B. They associated elaborations of phenomena in problem diagram with a data refinement in Event-B. The work of [5] proposed an iterative process of requirement modeling and validation. The authors connected reasoning about artifacts with refinement in Event-B. In [13], Event-B specifications are derived from class and state-machine diagrams. However, refinement strategy planning is not covered in these studies.

There have been many studies which aim at deriving formal specification in other methods than Event-B from natural language specifications or diagrams like UML [3,4,6–8,10,12,14,15] but refinement is not considered in these studies either.

The authors of [11] proposed a method to derive an abstract specification of an event. In this study, patterns of correspondences between a KAOS goal model and an event in Event-B are provided. The patterns also consider a part of proof obligations that will be generated. From the point of view of refinement strategy planning, this method transforms a refinement strategy planning for an event into a refinement strategy planning in a KAOS goal model. On the other hand, our method plans refinement strategy of the whole model by considering the constraints on introduction between elements in the system. Thus our approach can be considered as complementary to this work.

4.2 Future Work

Further refinement of the proposed method is the primary part of the future work.

First, we assumed every artifact is not changed through refinement. However, that is not the case in realistic situations. There are many cases that some artifacts are strengthened in concrete models by using newly introduced phenomena.

Refinement of events is also neglected. In realistic situations, many events are refined through refinements. For example, both event "Loaning a reserved books" and "Loaning a book from the open stack" can refine an event that only includes "Add one to loan state" state transition.

Moreover, as we viewed in Section 2, we assumed that the complexity of modeling can be measured only by the number of phenomena. However this criterion is too rough. For instance, the importance of the number of events and that of variables are different. Thus, finer analysis of complexity of modeling is needed.

5 Conclusion

This paper has aimed at resolving complexities in planning of refinement strategy by considering semantics constraints of Event-B. Refinement strategy planning is an important and difficult phase in modeling in Event-B. Therefore, the proposed method facilitates ordinary users to leverage Event-B. Although much work remains as discussed in Section 4.2, we believe this work promotes systematic use of formal specifications, more independently from specific knowledge and skills.

References

1. Formal methods application guidelines (in japanese). `http://sec.ipa.go.jp/reports/20120928.html`
2. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (May 2010)
3. Cabral, G., Sampaio, A.: Automated Formal Specification Generation and Refinement from Requirement Documents. Journal of the Brazilian Computer Society 14(1), 87–106 (2008)
4. Elbendak, M., Vickers, P., Rossiter, N.: Parsed Use Case Descriptions as a Basis for Object-Oriented Class Model Generation. Journal of Systems and Software 84(7), 1209–1223 (Jul 2011)
5. Hallerstede, S., Jastram, M., Ladenberger, L.: A method and tool for tracing requirements into specifications. in Electronic Communications of the EASST (2012)
6. Ilic, D.: Deriving formal specifications from informal requirements. In: Annual International Computer Software and Applications Conference, 2007. pp. 145–152 (2007)
7. Jin, L., Zhu, H.: Automatic Generation of Formal Specification from Requirements Definition. In: Formal Engineering Methods., 1997. pp. 243–251. No. 25 (1997)
8. Lausdahl, K., Lintrup, H.K., Larsen, P.G.: Connecting UML and VDM++ with open tool support. In: The 16th International Symposium on Formal Methods (FM 2009). pp. 563–578 (2009)
9. Loesch, F., Gmehlich, R., Grau, K., Jones, C., Mazzara, M.: Report on pilot deployment in automotive sector. Deliverable D7 (2010)
10. Macias, B., Pulmal, S.G.: A Method for Controlling the Production of Specifications in Natural Language. The Computer Journal 38(4), 310–318 (1995)
11. Matoussi, A., Gervais, F., Laleau, R.: A goal-based approach to guide the design of an abstract event-b specification. In: Engineering of Complex Computer Systems (ICECCS), 2011 16th IEEE International Conference on. pp. 139–148. IEEE (2011)
12. Miriyala, K., Harandi, M.T.: Automatic Derivation of Formal Software Specifications From Informal Descriptions. IEEE Transactions on Software Engineering 17(10), 1126–1142 (1991)
13. Said, M.Y., Butler, M., Snook, C.: Language and tool support for class and state machine refinement in uml-b. In: The 16th International Symposium on Formal Methods (FM 2009). pp. 579–595 (February 2009)
14. Vadera, S., Meziane, F.: From English to Formal Specifications. The Computer Journal 37(9), 753–763 (1994)
15. Wang, X., Liu, S., Miao, H.: A Pattern System to Support Refining Informal Ideas into Formal Expressions. In: Formal Methods and Software Engineering. pp. 662–677 (2010)